CS 595: Assignment #9

Due on Thursday, December 4, 2014

Dr Nelson 4:20PM

Victor Nwala

Contents

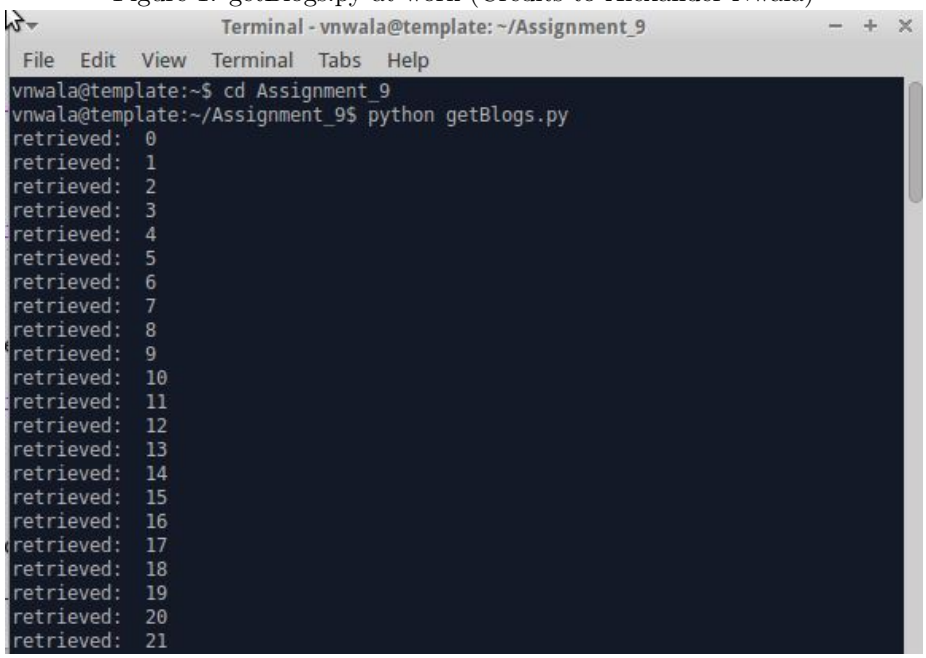
Problem 1

1. Create a blog-term matrix. Start by grabbing 100 blogs; include:
http://f-measure.blogspot.com/ http://ws-dl.blogspot.com/
and grab 98 more as per the method shown in class.

Listing 1: Code to extract links

```
import requests
import urlparse

def getUniqueBlogs(countOfBlogsToRetrieve):

     listOfBlogs = []
     if( countOfBlogsToRetrieve>0 ):

          try:
               outputFile = open('blogs.txt', 'w')
               outputFile.write('http://f-measure.blogspot.com/\n')
               outputFile.write('http://ws-dl.blogspot.com/\n')
          except:
               exc_type, exc_obj, exc_tb = sys.exc_info()
               fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
               print(fname, exc_tb.tb_lineno, sys.exc_info() )

          while len(listOfBlogs) != countOfBlogsToRetrieve:

               print 'retrieved: ', len(listOfBlogs)

               blogUrl = 'http://www.blogger.com/next-blog?navBar=true&blogID
                   =3471633091411211117'
               response = ''
               try:
                    response = requests.head(blogUrl, allow_redirects=True)
                    response = response.url
               except:
                    response = ''


               if len(response) > 0:
                    if response not in listOfBlogs:

                         response = response.lower()

                         parsedUrl = urlparse.urlparse(response)
                         parsedUrl = parsedUrl.scheme + '://' + parsedUrl.netloc +
                              '/'

                         listOfBlogs.append(parsedUrl)
                         outputFile.write(parsedUrl + '\n')

          outputFile.close()

getUniqueBlogs(98)
```

Figure 1: getBlogs.py at work (Credits to Alexander Nwala)



Listing 2: Code to get frequent 500 terms and create blogdata

```
import sys, os
import feedparser
import re
import math

def getwords(html):
    # Remove all the HTML tags
    txt = re.compile(r'<[^>]+>').sub('', html)

    # Split words by all non-alpha characters
    words = re.compile(r'[^A-Z^a-z]+').split(txt)

    # Convert to lowercase
    return [word.lower() for word in words if word != '']

def getwordcounts(url):

     '''
     Returns title and dictionary of word counts for an RSS feed
     '''
     # Parse the feed


     #url: http://blogName.blogspot.com/
     d = feedparser.parse(url)


     wc = {}

     # Loop over all the entries
```

```
    for e in d.entries:
        if 'summary' in e:
            summary = e.summary
        else:
            summary = e.description

        # Extract a list of words
        words = getwords(e.title + ' ' + summary)
        for word in words:
            wc.setdefault(word, 0)
            wc[word] += 1

    return (d.feed.title, wc)

def getFeedVector500Popular(blogsFilename):

    apcount = {}
    wordcounts = {}
    feedlist = [line for line in file(blogsFilename)]



    for feedurl in feedlist:

        feedurl = feedurl.strip()

        try:

            (title, wc) = getwordcounts(feedurl + 'feeds/posts/default?max-results=500')
            wordcounts[title] = wc
            for (word, count) in wc.items():
                apcount.setdefault(word, 0)
                if count > 1:
                    apcount[word] += 1

        except:
            print 'Failed to parse feed %s' % feedurl
            exc_type, exc_obj, exc_tb = sys.exc_info()
            fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
            print(fname, exc_tb.tb_lineno, sys.exc_info() )



    wordlist = []
    listOfTermTermFrequencyTuple = []
    for (t, tF) in apcount.items():

        frac = float(tF) / len(feedlist)

        if frac > 0.1 and frac < 0.5:
            ttFTuple = (t, tF)
            listOfTermTermFrequencyTuple.append(ttFTuple)
```

```
    #frequent 500 terms
    listOfTermTermFrequencyTuple = sorted(listOfTermTermFrequencyTuple, key=
        lambda tuple: tuple[1], reverse=True)
    for ttF in  listOfTermTermFrequencyTuple:
         t = ttF[0]
         tF = ttF[1]

         if( len(wordlist) > 500 ):
              break
         else:
              wordlist.append(t)


    out = file('blogVectorResult.txt', 'w')
    out.write('Blog')

    for word in wordlist:
         word = word.encode('ascii', 'ignore')
         out.write('\t%s' % word)

    out.write('\n')
    for (blog, wc) in wordcounts.items():

         blog = blog.encode('ascii', 'ignore')
         #print blog
         out.write(blog)
         for word in wordlist:

              word = word.encode('ascii', 'ignore')
              if word in wc:
                   out.write('\t%d' % wc[word])
              else:
                   out.write('\t0')
         out.write('\n')

def getFeedVector500PopularTFIDF(blogsFilename):

    apcount = {}
    wordcounts = {}
    feedlist = [line for line in file(blogsFilename)]



    for feedurl in feedlist:

         feedurl = feedurl.strip()

         try:

              (title, wc) = getwordcounts(feedurl + 'feeds/posts/default?max-
                  results=500')
```

```
                wordcounts[title] = wc
                for (word, count) in wc.items():
                     apcount.setdefault(word, 0)
                     if count > 1:
                          apcount[word] += 1

           except:
                print 'Failed to parse feed %s' % feedurl
                exc_type, exc_obj, exc_tb = sys.exc_info()
                fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
                print(fname, exc_tb.tb_lineno, sys.exc_info() )



      wordlist = []
      listOfTermTermFrequencyTuple = []
      for (t, tF) in apcount.items():

           frac = float(tF) / len(feedlist)

           if frac > 0.1 and frac < 0.5:
                ttFTuple = (t, tF)
                listOfTermTermFrequencyTuple.append(ttFTuple)


      #frequent 500 terms
      listOfTermTermFrequencyTuple = sorted(listOfTermTermFrequencyTuple, key=
          lambda tuple: tuple[1], reverse=True)
      for ttF in  listOfTermTermFrequencyTuple:
           t = ttF[0]
           tF = ttF[1]

           if( len(wordlist) > 500 ):
                break
           else:
                wordlist.append(t)


      out = file('blogVectorResultTFIDF.txt', 'w')
      out.write('Blog')

      for word in wordlist:
           word = word.encode('ascii', 'ignore')
           out.write('\t%s' % word)

      out.write('\n')
      for (blog, wc) in wordcounts.items():

           blog = blog.encode('ascii', 'ignore')
           #print blog
           out.write(blog)
           for word in wordlist:

```

```
                word = word.encode('ascii', 'ignore')
                if word in wc:

                    TF = wc[word]/float(len(wc))
                    IDF = len(feedlist)/float(apcount[word])

                    #log base 2:
                    IDF = math.log(IDF) / float(math.log(2))
                    TFIDF = TF*IDF

                    out.write('\t%f' % TFIDF)
                else:
                    out.write('\t0')
            out.write('\n')


#p1
#getFeedVector500Popular('blogs.txt')

#p5
getFeedVector500PopularTFIDF('blogs.txt')
```

Note:This code is used for both problem 1 and 5

Create a histogram of how many pages each blog has (e.g., 30 blogs with just one page, 27 with two pages, 29 with 3 pages and so on).

Listing 3: Code to count number of pages in a link

```
from bs4 import BeautifulSoup
import requests
import os, sys


def listOfPageFeedsPageCount(blog):

     feedPages = []
     if( len(blog) > 0 ):

          if( blog[len(blog)-1] != '/' ):
               blog = blog + '/feeds/posts/default/'
          else:
               blog = blog + 'feeds/posts/default/'

          listOfPageFeeds = paginate(blog, feedPages)


     return listOfPageFeeds

def paginate(blog, listOfPages=[]):

     html = requests.get(blog)
     link = BeautifulSoup(html.text).find('link', { 'rel' : 'next' })

     if link is not None:
          link = link['href']
          listOfPages.append(link)

          paginate(link, listOfPages)

     return listOfPages
out = open('numberOfPages.txt', 'a')

with open("blogs.txt", "r") as f:
  for line in f:
     url = line.strip()

     arrayOfPages = listOfPageFeedsPageCount(url)
     Number = len(arrayOfPages) + 1
     print Number
     out.write(str(Number))
     out.write('\n')
     for page in arrayOfPages:
          print page
out.close()
```

Figure 2: getNextPage.py at work (Credits to Alexander Nwala)

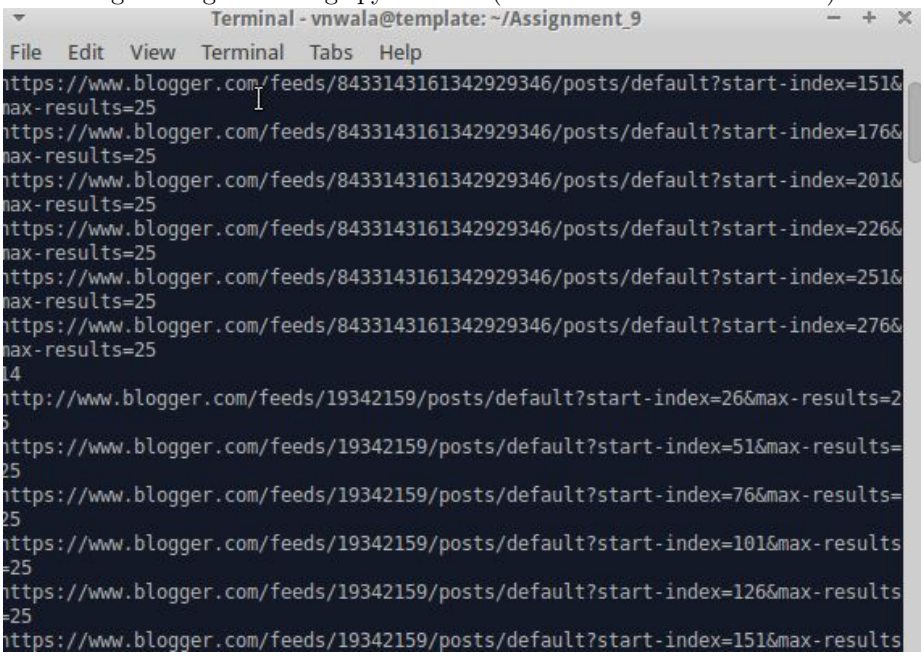
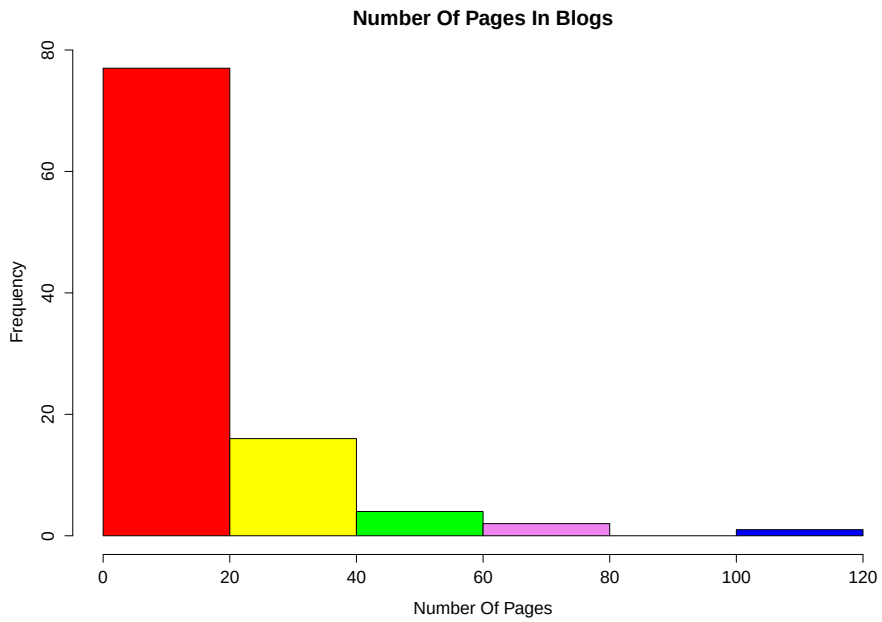


Figure 3: Histogram of Pages in Blogs



Problem 2

2. Create an ASCII and JPEG dendrogram that clusters (i.e., HAC) the most similar blogs (see slides 12 & 13). Include the JPEG in your report and upload the ascii file to github (it will be too unwieldy for inclusion in the report). The ASCII file is saved with the file name ascii.txt

Listing 4: Code to answer problem 2 to 5

```
import clusters
blognames,words,data=clusters.readfile('blogVectorResult.txt')
clust=clusters.hcluster(data)
#clusters.printclust(clust,labels=blognames)
reload(clusters)
#clusters.drawdendrogram(clust,blognames,jpeg='blogclust.jpg')
#kclust=clusters.kcluster(data,k=20)
coords=clusters.scaledown(data)
clusters.draw2d(coords,blognames,jpeg='blogs2d.jpg')
```

Figure 4: Ascii of most similar Blogs

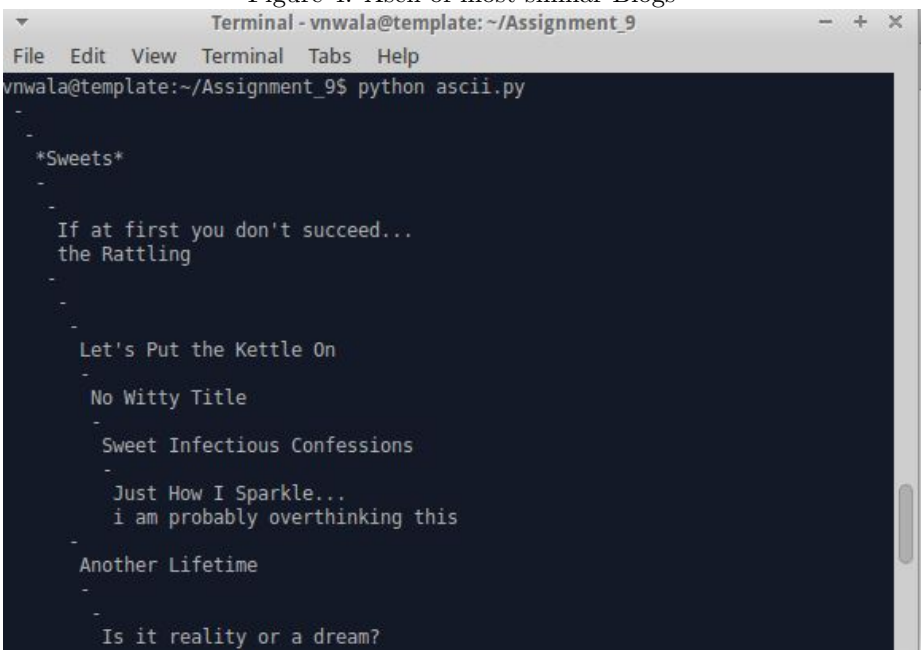
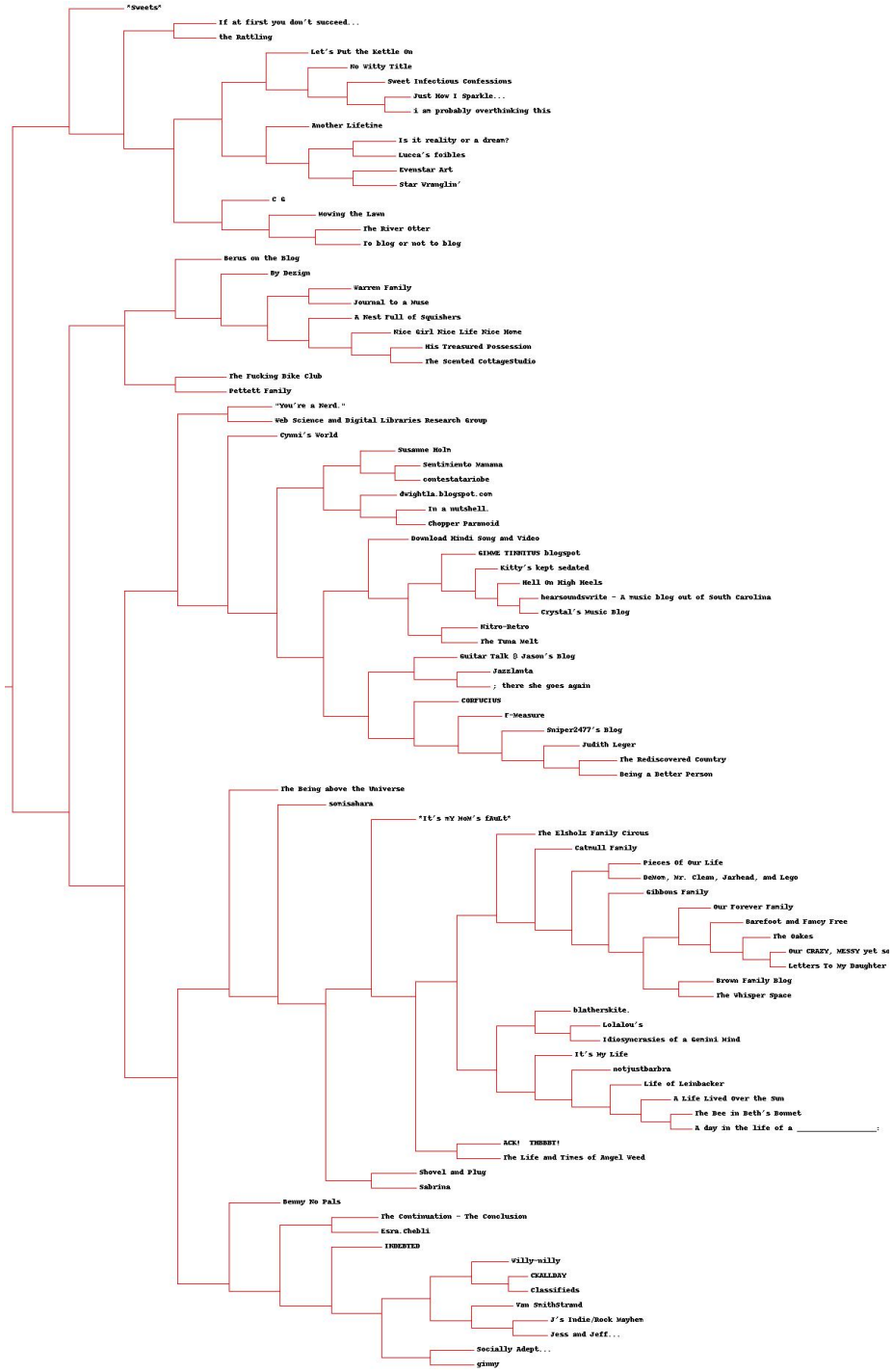


Figure 5: Dendogram for Blog



Problem 3

3. Cluster the blogs using K-Means, using k=5,10,20. (see slide 18). How many interations were required for each value of k? K = 5 has 8 iterations, k = 10 has 6 iterations while k = 20 has 5 iterations.

Figure 6: Showing number of iterations for K=5

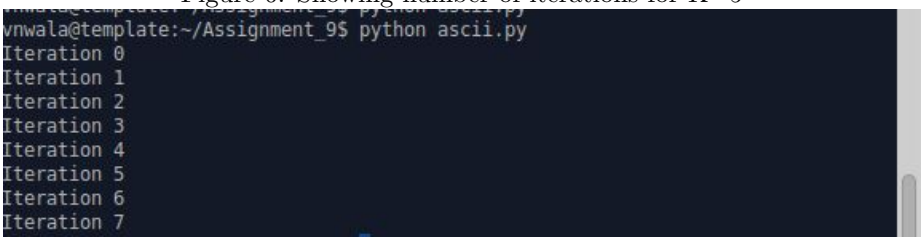


Figure 7: Showing number of iterations for K=10

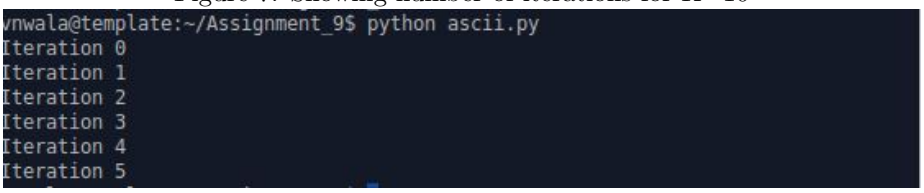
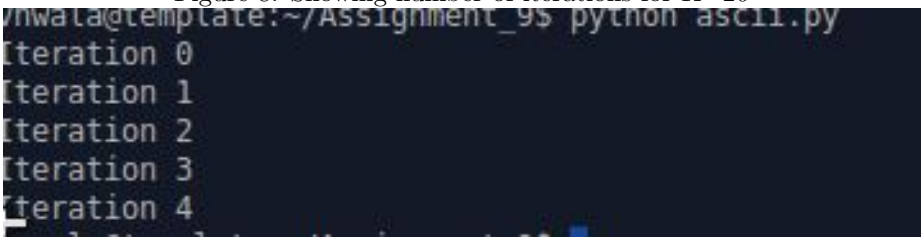


Figure 8: Showing number of iterations for K=20



Problem 4

4. Use MDS to create a JPEG of the blogs similar to slide 29. How many iterations were required?
To create this mds 4 iterations were required.

Figure 9: MDS JPEG of the Blogs

Figure 10: Showing number of iterations

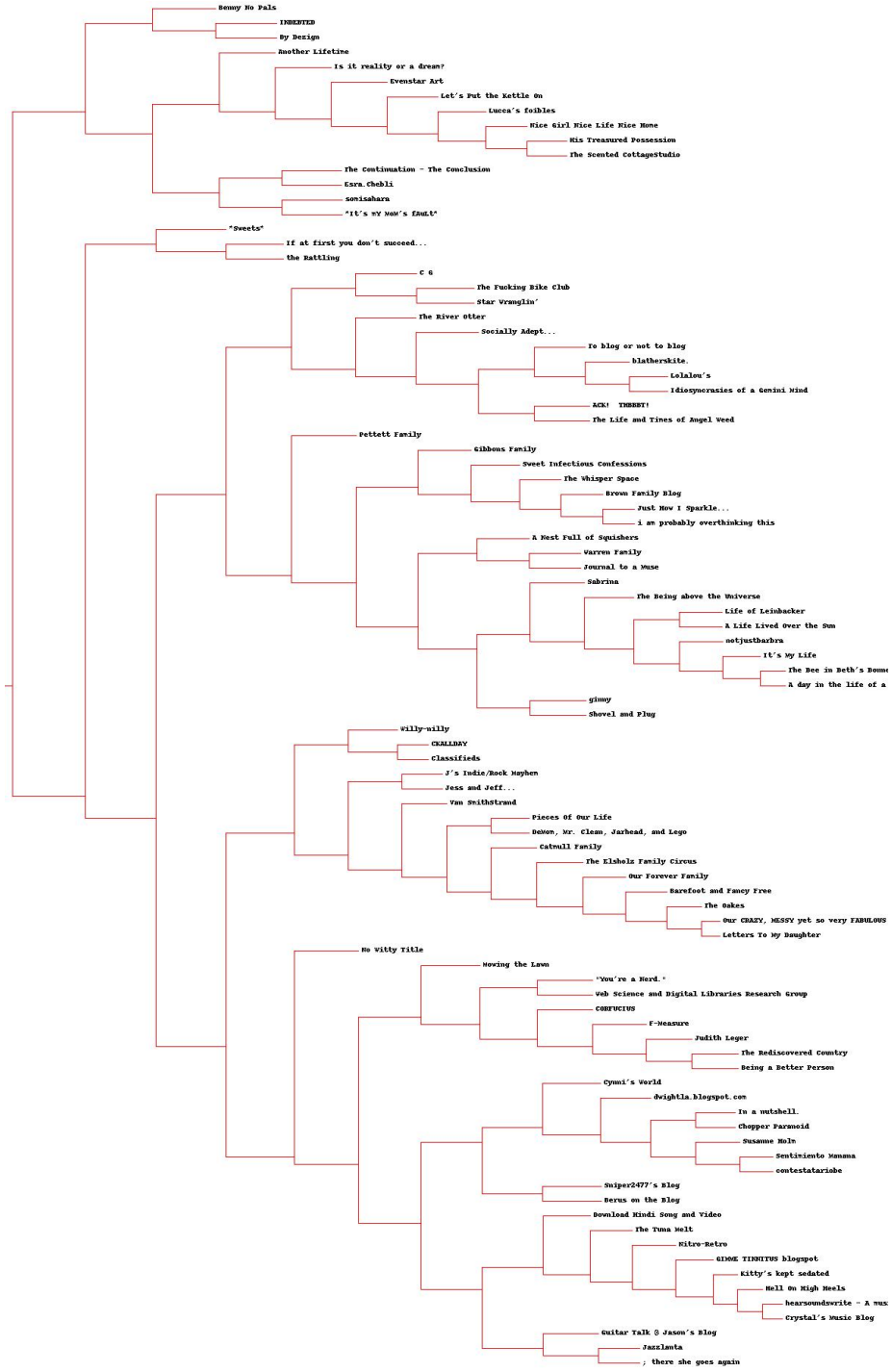
```
vnwala@template:~/Assignment_9$ python ascii.py
4167.20217831
4136.98910241
4131.88590136
4132.31654601
vnwala@template:~/Assignment_9$
```

Problem 5

5. Re-run question 2, but this time with proper TFIDF calculations instead of the hack discussed on slide 7 (p. 32). Use the same 500 words, but this time replace their frequency count with TFIDF scores as computed in assignment #3. Document the code, techniques, methods, etc. used to generate these TFIDF values. Upload the new data file to github.
Compare and contrast the resulting dendrogram with the dendrogram from question #2.
Note: ideally you would not reuse the same 500 terms and instead come up with TFIDF scores for all the terms and then choose the top 500 from that list, but I'm trying to limit the amount of work necessary.

Problem 5 continued on next page. . .

Figure 11: Showing Dendogram for Blog matrix generated form TFIDF calculations



The function getFeedVector500PopularTFIDF in get500Popular.py is responsible for calculating the TFIDF for the terms (words) in blog. To calculate the TF, number of times a word occurs is divided by the total number of words in that blog (Line 189 on get500Popular.py). For the IDF the logarithm to the base of two is taken of the division of the total number of words in the entire file (combination of 100 blogs) by the number of times the word of term occurs in the entire file (blog-matrix)(Line 190 and 193 on get500Popular.py). TFIDF is simply a multiplication of both results for a term (word)(Line 194 on get500Popular.py).

Comparing and Contrasting the dendograms generated from TFIDF calculation and those generated from

question 2, I observed certain blogs titles appeared in the same cluster in both cases. For example, "Sweets", "If at first you don't succeed...", "the Rattling" appeared in the same cluster in both cases, others who followed the same trend are, "Willy-nilly","CKALLDAY","Classifieds","Van SmithStrand","J's Indie/Rock Mayhem", "Jess and Jeff...", others are, "You're a Nerd" and "Web Science and Digital Libraries Research Group", finally "CORFUCIUS" and "F-Measure" . These are some similarites I observed. Also the ASCII file generated in problem 5 is saved with the name asciiTFIDF.txt

Conclusion

To conclude, I should state that Alexander Nwala was a huge contributor to solving problem 1 and 5.

References

[1] arthur e. Programming-collective-intelligence. https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter3/generatefeedvector.py, 24 Decenber 2012.

[2] arthur e. Programming-collective-intelligence. https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter3/clusters.py, 24 Decenber 2012.

[3] Toby Segaran. *Programming Collective Intelligence*. Safari, August 2007.

[]